Future Generation Computer Systems 79 (2018) 786–796

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Efficient CFD code implementation for the ARM-based Mont-Blanc architecture

G. Oyarzun [a,b,*], R. Borrell [a,b], A. Gorobets [b,c], F. Mantovani [d], A. Oliva [b]

[a] Termo Fluids, S.L., c/ Magí Colet 8, 08204 Sabadell (Barcelona), Spain
[b] Heat and Mass Transfer Technological Center, ETSEIAT, Technical University of Catalonia, C/ Colom 11, 08222, Terrassa, Spain
[c] Keldysh Institute of Applied Mathematics RAS, 4A, Miusskaya Sq., Moscow, 125047, Russia
[d] Barcelona Supercomputing Center, c/ Jordi Girona 3, 08034, Barcelona, Spain

HIGHLIGHTS

- Termo Fluids CFD code has been run on up to 128 ARM-based Mont-Blanc nodes.
- An heterogeneous implementation has been developed to occupy the overall system.
- A dynamic Tabu search load balance algorithm distributes the workload among devices.
- The hybrid approach is up to times faster than the CPU-only version of the code.
- Mont-Blanc nodes are 41% more energy efficient than Minotauro hybrid nodes.

ARTICLE INFO

Article history:
Received 20 April 2017
Received in revised form 9 August 2017
Accepted 10 September 2017
Available online 20 September 2017

Keywords:
ARM system
Heterogeneous computing
Parallel CFD
Energy-efficient computing

ABSTRACT

Since 2011, the European project Mont-Blanc has been focused on enabling ARM-based technology for HPC, developing both hardware platforms and system software. The latest Mont-Blanc prototypes use system-on-chip (SoC) devices that combine a CPU and a GPU sharing a common main memory. Specific developments of parallel computing software and well-suited implementation approaches are of crucial importance for such a heterogeneous architecture in order to efficiently exploit its potential.

This paper is devoted to the optimizations carried out in the TermoFluids CFD code to efficiently run it on the Mont-Blanc system. The underlying numerical method is based on an unstructured finite-volume discretization of the Navier–Stokes equations for the numerical simulation of incompressible turbulent flows. It is implemented using a portable and modular operational approach based on a minimal set of linear algebra operations. An architecture-specific heterogeneous multilevel MPI+OpenMP+OpenCL implementation of such kernels is proposed. It includes optimizations of the storage formats, dynamic load balancing between the CPU and GPU devices and hiding of communication overheads by overlapping computations and data transfers. A detailed performance study shows time reductions of up to 2.1× on the kernels' execution with the new heterogeneous implementation, its scalability on up to 128 Mont-Blanc nodes and the energy savings (around 40%) achieved with the Mont-Blanc system versus the high-end hybrid supercomputer MinoTauro.

1. Introduction

Exascale computing (10^{18} floating-point operations per second) is the next major milestone in the process of exponential growth of HPC systems occurred for over half a century (the Moore's law) [1]. One of the main concerns of the HPC community is that current high-end hardware cannot achieve this leap with reasonable power consumption (≈ 20MW) [2]. Nowadays, several public and private institutions around the world are investigating different aspects that should lead to the future generation of supercomputers, looking for innovative solutions in the way supercomputers interconnect, compute and move and store data. This set of initiatives have entitled this problem as the Exascale challenge.

In the road to Exascale computing, the Mont-Blanc project [3] is a European initiative devoted to design a new type of HPC systems built from low-power commercially available embedded technology. The idea consists in leveraging the huge investments on the exponentially growing market of mobile devices. The hybrid nodes of the Mont-Blanc prototype contain ARM-based CPUs

* Corresponding author at: Heat and Mass Transfer Technological Center, ETSEIAT, Technical University of Catalonia, C/ Colom 11, 08222, Terrassa, Spain.
E-mail address: guillermo@cttc.upc.edu (G. Oyarzun).

http://dx.doi.org/10.1016/j.future.2017.09.029

and GPUs fused in a System-on-Chip (SoC) architecture. This low-power consumption architecture has the potential of becoming an interesting cost-effective solution for future HPC-systems [4]. Nonetheless, the competitiveness of such an architecture relies upon the development of algorithms and software that fully exploit its capabilities. In this context, new tools like Climbing Mont Blanc (CMB) [5] have been created in order to facilitating the training and competitions in energy efficient programming of state-of-the-art heterogeneous multicores.

The initial prototype of the Mont-Blanc project was named Tibidabo [6], and consisted of 128 nodes equipped with the NVIDIA Tegra 2 device which includes a GPU and the dual-core ARM Cortex-A9 CPU. The Tegra 2 GPU did not support general purpose computing; therefore, the first experiences were restricted to the CPUs [7,8]. The nodes interconnection and its limitations were studied in [9]. One of the first attempts of porting CFD applications to mobile architectures was presented in [10], where scalability tests were presented using up to 96 nodes, while in [11] it is presented a study of the energy efficiency of a CFD code on embedded platforms. In the current version the Mont-Blanc prototype is composed of 930 nodes with the SoC Samsung Exynos 5 that combines an ARM Cortex-A15 CPU and an OpenCL capable Mali T604 GPU. The description of the hardware and general performance results can be found in [12]. A comparative study of the performance of the two devices composing each chip can be found in [13]. Another example is [14], where an OpenCL implementation of an N-body algorithm for mobile GPUs is presented.

While it is true that the Mont-Blanc system provides the necessary tools to run a numerical application, specific adaptations are required in order to exploit the resources efficiently. In this paper we present the architecture-specific developments carried out in TermoFluids (TF) code. TF is a multi-physics CFD code developed by Termo Fluids S.L. and its partners [15,16], which has been used in many industrial and academic numerical studies. The present work was carried out in the context of the End User Group of the Mont-Blanc 2 project.

The implementation approach of TF has been recently evolved to improve its portability across different architectures. In particular, the time integration phase has been re-designed to base it only on the composition of three linear algebra kernels: the sparse-matrix vector product (SpMV), the linear combination of two vectors (referred as AXPY in the BLAS standard [17]), and the dot product. With this approach the non-linear operators, such as the convective term, are expressed as the concatenation of two SpMV. Further details can be found in [18]. This constitutes the baseline of the present paper, which is focused on optimizing the basic kernels composing the time integration to efficiently run on the Mont-Blanc prototype. In contrast to previous works [10,13], where applications were only ported to run on one of the two devices composing the Samsung Exynos 5 chip, here we present a multilevel heterogeneous approach that allows the concurrent execution in both the CPU and GPU devices.

A domain decomposition is used to distribute the problem among different nodes, being data transfers implemented through the Message Passing Interface (MPI) standard. The costs of those transfers are partially hidden (up to 67%) by using an overlapping strategy that allows performing computations and communications simultaneously. A shared memory OpenMP parallelization is used within the multi-core CPUs. The OpenCL computing standard is used for the GPUs. Finally, a load-balancing algorithm based on a Tabu search strategy is used to distribute the load among devices within each node. The new adapted heterogeneous implementation delivers around 1.35× speedup on average compared to the baseline CFD algorithm suitable for a CPU-only execution with single-level MPI parallelization. A detailed performance study, presented in Section 5, shows time reductions on the kernels' execution of up to 2.1× with the new heterogeneous implementation, its scalability on up to 128 Mont-Blanc nodes and the energy savings (around 40%) achieved with the Mont-Blanc system versus a high-end hybrid supercomputer MinoTauro.

The rest of this paper is organized as follows: in Section 2 the mathematical model and its implementation are shortly described. Details about the Mont-Blanc prototype and the programming model are given in Section 3. Section 4 is devoted to the adaptation and optimization of the CFD algorithm. The performance study is presented in Section 5 together with an analysis of the energy costs. Finally, Section 6 is devoted to concluding remarks.

2. Governing equations and numerical method

The simulation of a turbulent flow of an incompressible Newtonian fluid is considered. The flow field is described by the dimensionless incompressible Navier–Stokes (NS) equations and the temperature transport equation:

$$\nabla \cdot \mathbf{u} = 0, \tag{1}$$

$$\partial_t \mathbf{u} + C(\mathbf{u}, \mathbf{u}) = \mathcal{D}\mathbf{u} - \nabla p + \boldsymbol{f}, \tag{2}$$

$$\partial_t \theta + C(\mathbf{u}, \theta) = Pr^{-1}\mathcal{D}\theta, \tag{3}$$

where $\mathbf{u}$ is the 3D velocity vector, θ is the temperature, and the convective term is $C(\mathbf{u}, \phi) = (\mathbf{u} \cdot \nabla)\phi$. In the forced convection case, with density variations neglected the temperature is a passive scalar, the body force vector is $\boldsymbol{f} = 0$, and the diffusive term reads $\mathcal{D}\mathbf{u} = Re^{-1}\nabla^2\mathbf{u}$, where Re is the Reynolds number. In the case of natural convection, when the density variations are not neglected, the diffusive term becomes $\mathcal{D}\mathbf{u} = PrRa^{-1/2}\nabla^2\mathbf{u}$, and $\boldsymbol{f} = (0, 0, Pr\theta)$ (Boussinesq approximation), where Ra and Pr are the Rayleigh and Prandtl numbers, respectively.

The NS equations (2) are discretized using a cell-centered symmetry-preserving discretization [19] on a collocated unstructured mesh. The operator-based finite-volume spatial discretization of the equations reads

$$\Omega \frac{d\mathbf{u}_h}{dt} + \mathsf{C}(\mathbf{u}_h)\,\mathbf{u}_h + \mathsf{D}\mathbf{u}_h - \mathsf{M}^t\boldsymbol{p}_h = 0, \tag{4}$$

where the discrete incompressibility constraint is given by $\mathsf{M}\mathbf{u}_h = 0$. The diffusive matrix, D, is symmetric and positive semi-definite. It represents the integral of the diffusive flux through the faces. The diagonal matrix, Ω, describes the sizes of control volumes. The approximate convective flux is discretized as in [19]. The time evolution of the temperature, θ_h, is discretized similarly. A second-order explicit one-leg scheme is used for the temporal discretization of both the convective and diffusive terms. The pressure–velocity coupling is solved with the classical fractional step projection method [20]. Consequently, a Poisson equation for $\boldsymbol{p}_h^{n+1}$ is solved on each time-step,

$$\mathsf{L}\boldsymbol{p}_h^{n+1} = \mathsf{M}\mathbf{u}_h^{p} \qquad \text{with} \qquad \mathsf{L} = -\mathsf{M}\Omega^{-1}\mathsf{M}^t, \tag{5}$$

where the discrete Laplacian operator, L, is represented by a symmetric negative semi-definite matrix.

Our implementation of this CFD solver is based on the algebraic operational approach described in [18]. Basically, we replaced the stencil data structures and the sweeps through mesh elements by sparse matrices and sparse-matrix vector products (SpMV), respectively. Such an approach has higher modularity without sacrificing performance and consequently can be easily ported to any architecture. Sparse matrices are stored in compressed formats. The matrix entries are saved in a one-dimensional double precision array and additional integer arrays are included in order to describe the sparsity pattern, i.e. position of each entry in the matrix.

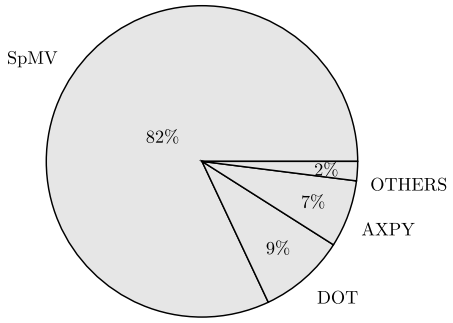

Fig. 1. LES simulation of flow around ASMO car. Mesh 5.5M. Execution using 32 GPUs. Distribution of computing time among the main kernels on the time-integration.

On the one hand, the linear operators that remain constant during all the simulation, such as the gradient and the divergence, can be directly represented by a sparse matrix and are applied through a SpMV. On the other hand, the non-linear operators, such as the convective term, need to be reevaluated at each time step. If the mesh topology does not change the sparsity pattern of the matrix remains constant, therefore only the one-dimensional array of matrix entries needs to be updated. In this case the nonlinear convective operator is implemented as a concatenation of two SpMVs. The first SpMV takes the advection field as input and results in the vector of updated coefficients (this array of matrix coefficients is treated as a vector). Then, the second SpMV applies the updated matrix to the input vector of the convection operator. Note that the definition of the first operator depends on the storage format used for the convective operator. An example for the sliced ELLPACK format is shown in [21].

The preconditioned conjugate gradient solver is used for the Poisson equation. Either the Jacobi diagonal scaling or the approximate inverse preconditioner are used depending on each particular case. Both preconditioners can be represented as a SpMV at the solution stage . Further details can be found in [22]. In conclusion, we express the full time integration step in terms of three algebraic kernels: the SpMV, the linear combination of two vectors (AXPY) and the dot product.

Fig. 1, exemplifies the operational approach for a Large Eddie Simulation (LES) of the flow around ASMO car performed on a hybrid cluster with 128 6-core Intel Westmere E5649 CPUs and 128 NVIDIA M2090 GPUs. The 98% of the computing costs are spent on the three main algebraic kernels, being the SpMV the dominant one among them. This circumstance makes the code extremely portable to any architecture.

3. Mont-Blanc prototype

3.1. Computer system architecture

The Mont-Blanc project [3] is a European research project started in 2011 with the goal of leveraging the fast growing market of smartphones and tablets for performing parallel scientific computing. The project has deployed a fully functional large scale prototype based on mobile SoCs as well as the system software stack needed to operate it as on a standard HPC cluster. Currently, the Mont-Blanc cluster is composed of 930 prototype nodes that use ARM SoC technology adapted from the mobile industry. The primary features of the system nodes are:

- ARM Cortex-A15, a dual core RISC CPU at 1.7 GHz with 128-bit SIMD extensions, 64 KB of L1 cache and 1 MB of L2 cache.
- ARM Mali T604 GPU with four stream multiprocessors, each one supporting up to 256 active threads. Its 128-bit vector registers can fit two double precision real values.
- 4 GB of LPDDR3-1600 RAM memory that is physically shared between the GPU and the CPU devices. Both devices share a common memory bandwidth of 12.8 GB/s.
- The nodes are interconnected via the ASIX AS88179 USB 3.0 to 1 Gb Ethernet bridge, and an Ethernet PHY.
- The local storage of the node is limited to a 16 GB uSD card.

These low-end computing units provide a significantly lower performance than the high-end devices of traditional systems. However, the mobile SoCs have a much lower price and a reduced power consumption. Moreover, this architecture allows the CPU and GPU devices to physically share a common memory space. This feature can be viewed as a step forward in the evolution of heterogeneous systems, since it eliminates the communication overhead that comes from the data transfer between the CPU and GPU devices. Similar solutions providing a higher level of integration are likely to be found in future generations of heterogeneous systems. Those novel aspects of the Mont-Blanc nodes have motivated us to explore the development of a concurrent execution model capable of attaining the maximum performance.

3.2. Programming model

The different components of the system are utilized by means of a multilevel parallel model that combines three layers of parallelism: distributed memory MIMD parallelization by means of MPI 3.0 standard couples multiple hybrid nodes, shared memory MIMD parallelization by means of OpenMP 3.0 standard engages multicore CPUs, stream processing and SIMD parallelism by means of OpenCL 1.1 standard engages GPUs (see Fig. 2).

The choice of the frameworks is governed by the node components. The GPU only supports OpenCL, so CUDA and its libraries could not be used. For the CPU there were no OpenCL drivers available, so OpenMP is used instead. Compilation of the code is performed using the GNU C++ compiler. It contains the necessary extensions for the ARM architecture (additional flags are the architecture flag -mcpu=cortex-a15 and double precision flags -mfpu=vfpv4 -mfloat-abi=hard).

The common memory of the CPU and GPU is used by allocating regions in memory (buffers) accessible from both OpenCL kernels and OpenMP threads. Such specific buffers are allocated by means of the OpenCL API using the clCreateBuffer function with the flag CL_MEM_ALLOC_HOST_PTR. By doing so, a zero-copy mode is established when accessing the buffer from the CPU. This means that OpenMP threads do not create local copies of the memory region, but operate directly on it. The clEnqueueMapBuffer and clEnqueueUnmapMemObject functions are used for controlling the data accesses from the OpenMP threads.

For a concurrent execution, the master thread of the CPU sends to the GPU execution queue a non-blocking kernel execution by the OpenCL function clEnqueueNDRangeKernel. This operation launches the kernel on the GPU and, since it is non-blocking, returns to the execution queue of the CPU. Then, the master thread of the CPU launches the OpenMP version of the kernel. When the CPU has finished its work, the master thread creates a synchronization point with the GPU by means of the OpenCL function clFinish. This is a blocking operation that holds the execution until all the kernels in the GPU queue have finished their execution.

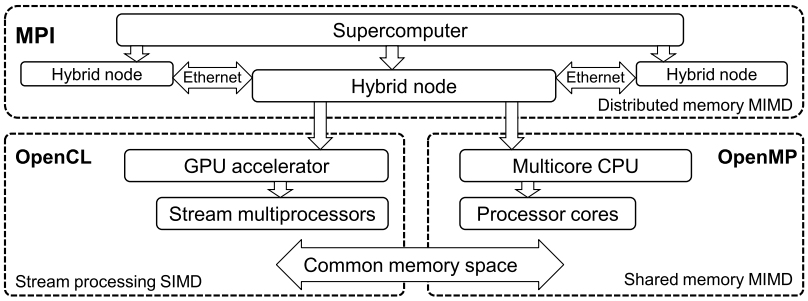

Fig. 2. Levels of parallelism and stack of frameworks for Mont-Blanc architecture.

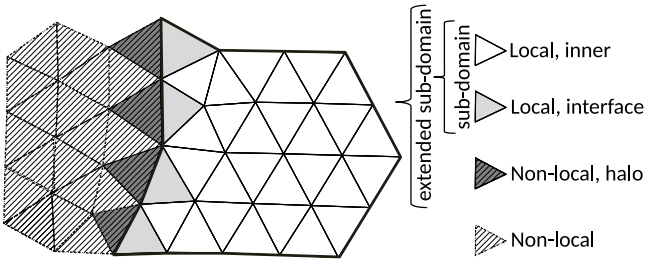

Fig. 3. Domain decomposition.

4. Algorithmic optimizations

Several issues must be addressed in order to use the Mont-Blanc prototypes efficiently. Firstly, a significant reduction of the inter-node data transfer overhead is possible with an overlap of communications and computations on the GPU devices. Secondly, intra-node optimizations that improve data layout and device-specific kernels are important. Finally, a proper dynamic load balancing between the CPU and GPU is required. Details on those aspects are presented in this section.

4.1. Inter-node communications

The first level of parallelization is a domain decomposition. Consequently the mesh $\mathcal{M}$, is decomposed into P non-overlapping subdomains $\mathcal{M}_0, \ldots, \mathcal{M}_{P-1}$. For each MPI process, cells of its subdomain are referred as *local* cells, likewise the corresponding unknowns are referred as local unknowns. A local cell adjacent to a non-local cell is denoted as *interface* cell, these form the inside subdomain border. The local cells that are not part of the interface are referred as *inner* cells. A non-local cell with an adjacent local cell is referred as a *halo* cell, these form the outside subdomain border. This decomposition is illustrated in Fig. 3.

The MPI communication patterns differ among the three main kernels. In the AXPY computations are performed on the local components of the vectors without any MPI communication episode. The dot product requires a global reduction operation after the local sum is calculated by each MPI process. This reduction communication has a cost of $O(log(P))$, where P is the number of MPI processes engaged. Finally, the SpMV requires a point-to-point communication such that each MPI-process obtains the halo components of the multiplying vector. This point-to-point communication is referred as a halo update. Note that the product of the inner components, and the halo update are independent operations that can be performed concurrently. The subsets of inner, interface and halo vector components and its associated matrix rows are illustrated in Fig. 4 (left). The overlapping process is illustrated in Fig. 4 (right). More details on these aspects can be found in [22].

4.2. Intra-device optimization

Two aspects must be taken into account for an optimal intra device performance: the data layout and the intra device parallelization. Both aspects are strongly related: an optimal data-layout is the one that minimizes the memory transfers on the parallel execution.

Regarding the vector operations there is not much discussion about the data layout, vectors are stored on 1D arrays. The OpenMP parallelization for the ARM Cortex-A15 CPU is done by distributing loop iterations between threads with work-sharing directives. Vectorization is not used since it is not supported by this CPU for double precision values. The four stream multiprocessors of the Mali T604 GPU are occupied with threads (work-items) launched by means of OpenCL. The SIMD registers of the GPU have a 128-bit width, thus two double precision operations can be performed at once. OpenCL vector data types: `double2` and `int2`, are used in order to optimize the vectorization.

Regarding the SpMV operation, the data layout (storage format) depends on the sparsity pattern of the matrix [23]. The most widely used format is known as Compressed Sparse Row (CSR). Such format consists in storing three separate arrays: the non-zero entries, the column indices and the pointers indicating the beginning and the end of each row, respectively. For matrices with equal number of entries per row, the vector of row pointers is redundant, by eliminating it we get the ELLPACK format. The regularity required by the ELLPACK format is forced by padding zeros in the rows with less entries. An optimized option to minimize the zero padding is the sliced ELLPACK (sELL) format. It consists in sorting the rows by the number of entries and then divide the matrix into slices which are themselves stored using the ELLPACK format. More details can be found in [18,21]. A performance comparison between the CSR and the sELL in our application context is presented in the next section.

The SpMV algorithm is composed of two loops: the outer loop iterates over the row indexes of the matrix and the inner loop across the non-zero entries of each row. In the ARM Cortex-A15 CPU the parallelization consist in using OpenMP directives for distributing the outer loop, assigning dynamically created chunks of rows to the OpenMP threads. In the Mali T604 GPU the stream processing model consists in launching thousands of threads (work-items) which are grouped in independent blocks (work-groups) dynamically scheduled to the stream-multiprocessors.

The data layout of the ELLPACK format in the main memory is influenced by the computing approach and differs among the two computing units. In the CPU the OpenMP threads process one row after the other, so a row-major order is used to optimize the memory accesses, see Fig. 5 (top). In the GPU the memory requests within a work-group are handled as a single transaction, i.e. all of its work-items process simultaneously the uploaded data. In this case each work-item operates two rows simultaneously to take advantage of the vectorization. If each work-group has M work items,

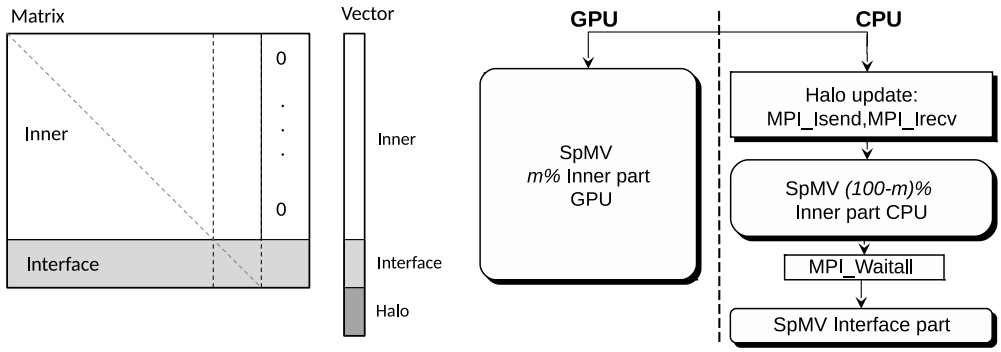

Fig. 4. Distributed matrix and vector structure (left) and concurrent execution model (right).

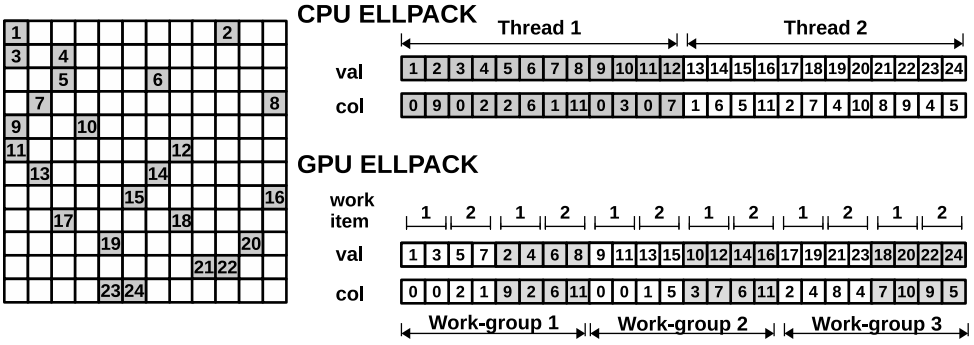

Fig. 5. Data layout of the ELLPACK format for the CPU and GPU devices.

the matrix rows are divided in blocks of size 2M and a column-wise order is prescribed within each block. This way when a column of one of those blocks is uploaded to the stream multiprocessor all of its threads can operate on it. This strategy is illustrated in Fig. 5 (bottom) for a hypothetical GPU with two threads (work-items) per stream-multiprocessor (workgroup). The Mali T604 GPU has 4 stream multiprocessors with 256 active threads each.

Additionally to the storage format, the SpMV performance depends on the indirect memory accesses required for the multiplying vector. These can be optimized with a matrix bandwidth reduction algorithm. The lower the matrix bandwidth the lower the distance between components used on the same row and, consequently, the better is exploited the spacial locality for cache reuse. In TF we re-order the vector and matrices components at three different levels: (i) firstly, in order to optimize communications, the components are grouped into the inner, the interface and the halo subgroups (illustrated in Fig. 4); (ii) within each of the previous subgroups, components are reordered by the number of entries of the respective row (to efficiently apply the sliced ELLPACK format); (iii) at the third level of the hierarchy (i.e. for the components corresponding to rows with the same number of entries and in the same subgroup with respect to the first criteria) we apply the Cuthill–McKee [24] bandwidth reduction algorithm. In this paper we consider static meshes therefore this reordering is only carried out at the preprocessing stage.

4.3. Load balancing inside hybrid nodes

An optimized heterogeneous implementation for such fused devices (where CPU and GPU compete for a shared memory bandwidth) is necessary to maximize the occupancy of the system. Our idea consists in finding for each kernel a distribution of workload between CPU and GPU that minimizes the execution time. This distribution is defined by the percentage of work, m%, assigned to GPU, and $(100 - m)$% to CPU, respectively. Since fused devices operate in a shared memory space, a specific distribution can be set without data transfer overhead.

The proposed load balancing optimization is based on a Tabu search strategy (outlined in Fig. 6), which finds the best distribution for each of the three main kernels. The Tabu search is a heuristic method for solving optimization processes avoiding sub-optimal solutions in an iterative process. When a solution m is found, it is perturbed with $\pm\Delta m$ and the iterative process is continued through the new candidates that have not been explored before, i.e. are not in the *Tabu list*. The iteration process continues until all the elements in the candidate list have been tested. Similar strategies using this method can be found in [25]. Note that for the SpMV kernel, in which computations and communications are overlapped, the communication costs must be included in the workload distribution, therefore, the result depends on the number of computing nodes used in the simulation.

An alternative approach for the SpMV distribution on hybrid systems can be found in [26,27]. Their strategy consists in using a probabilistic mass function to represent the distribution pattern of the non-zeros, and afterwards generate an hybrid storage format according to it.

5. Performance tests

The overall computing performance in a LES simulation is studied in terms of achieved GFLOPS rate and scalability. The numerical tests were executed on the Mont-Blanc nodes described in Section 3. In summary, each node has a SoC Samsung Exynos 5 that combines an ARM Cortex-A15 CPU and an OpenCL capable Mali T604 GPU. The nodes are interconnected by means of a Gigabit Ethernet network. The theoretical peak performance of the SoC is 28.1 GFLOPS and the memory bandwidth is 12.8 GB/s. We focus on the analysis of the three main kernels that compose our algorithm, since its wall-clock time sums up to 98% within the time integration phase.

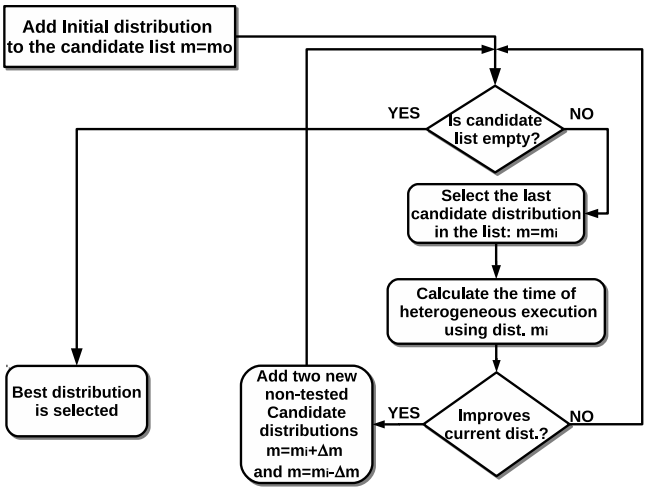

Fig. 6. Flow chart of the Tabu search algorithm for finding the best distribution.

Table 1
Performance in GFLOPS obtained with the CSR and sELL formats for meshes of different sizes.

Device, SpMV format	Mesh size, thousands of cells				
	50	100	200	400	800
CPU CSR	0.31	0.29	0.29	0.27	0.27
CPU sliced ELLPACK	**0.42**	**0.41**	**0.39**	**0.37**	**0.37**
GPU CSR	0.34	0.35	0.36	0.35	0.36
GPU sliced ELLPACK	**0.59**	**0.59**	**0.59**	**0.57**	**0.58**

5.1. Intra node performance: SpMV data layout

For the SpMV tests we have used a second order discretization of the Laplacian operator on tetrahedral-based meshes. The SpMV is a memory bounded operation. In ideal conditions, i.e. with maximal cache reuse for the input vector, the arithmetic intensity is only 9/76 FLOPS per byte, see [18] for details. Therefore the storage format is a major aspect to be considered in order to optimize the performance.

Table 1 shows the GFLOPS achieved at running the SpMV on the CPU and GPU of the Mont-Blanc nodes separately. We compare the CSR and sliced ELLPACK formats for meshes of different sizes. As explained in Section 4.2, using the ELLPACK format reduces the memory transfers and improves cache use since the vector of row indices is eliminated. Moreover, dealing with a constant number of entries per row eliminates the thread divergence[1] within workgroups in the GPU. Consequently, the sELL format outperforms the CSR with average speedups of 1.37× and 1.65× on the CPU and GPU devices, respectively. In all the following tests of the paper we use the sELL format, with the specific data layout for each device described in Section 4.2.

The AXPY and dot operations are also memory bounded kernels with arithmetic intensities of 1/12 and 1/8 FLOPS per byte, respectively. However in this case there is not possible optimization of the data layout which must be a one-dimensional array.

5.2. Intra node performance: Heterogeneous execution

In Fig. 7 the performance of the three main kernels for different mesh sizes is illustrated. Mont-Blanc nodes have 4 Gbytes of main memory; according to this we have done tests on a range of problem sizes for which performing a complete LES simulation would be reasonable. The performance in GFLOPS achieved for the separate executions on CPU and GPU is compared to the heterogeneous execution on both devices

The CPU performance is almost constant for each kernel. In particular, for the SPMV this means that we are beyond the range where cache effects take place. The performance ratio between the dot and AXPY is in average 1.33, similar to the ratio between their arithmetic intensities, 1.5. Note that the dot requires a reduction operation within the parallel region that degrades its performance. The SpMV has a higher arithmetic intensity than the AXPY but slightly lower performance. The reason are the random memory accesses of the input vector. Those cannot be well predicted by the prefetching units, converting the SpMV into a latency bounded kernel.

For the throughput-oriented GPU execution we observe a different behavior. In particular, the GPU design is based on launching large numbers of threads to the stream multiprocessors and using context switching to hide memory costs. This approach outperforms the latency-oriented design of the CPU, being especially noticeable for the SpMV. However, enough occupancy is required in order to saturate the device and achieve the maximal performance. This explains the performance growth on the vectorial kernels which under-occupy the device for the smaller problem sizes. In addition, we observe that despite the fact that the dot product has a higher arithmetic intensity it performs almost equally as the AXPY. This is because the reduction operation has very low performance on the GPU.

The benefits from the heterogeneous execution are not the same for the different kernels. For the bandwidth bounded vectorial kernels, having two devices generating memory requests at the same time produces relatively low improvements. This is because the bandwidth is already quite well saturated with a single device, especially with the GPU. For the SpMV the improvement is bigger. In average the speedup is 1.97× and 1.30× compared to the CPU and the GPU executions, respectively. In this case, the limitation are the latency costs derived from the cache misses produced by the indirect memory accesses. Therefore increasing the density of memory requests allows to better profit the highly under-saturated bandwidth. Note that the low arithmetic intensity of the main algebraic operations limits our maximum achievable performance to less than 3% of the theoretical peak, however this is in agreement with what we have observed on other heterogeneous systems for CFD applications [18].

Regarding the load balancing, as explained in Section 4.3, it is very hard to predict a good distribution a priori, due to the performance variability when CPU and GPU devices run concurrently or with the problem size. Consequently we use an iterative load balance optimization based on a Tabu search algorithm. In Fig. 8(left) the performance obtained with different load distributions is shown for the three main kernels, running a problem with 200K unknowns. Note that the performance suffers significant changes, what makes the load balance optimization a major issue. Fig. 8 (right) shows the optimal distribution achieved with the Tabu search algorithm for different problem sizes. For the SpMV we observe an almost flat line, the reason is the nearly constant performance of the two devices with the problem sizes. This optimal distribution is of around 70% of the load on the GPU, that in this kernel clearly outperforms the CPU. For the AXPY we see a moderate increase of the load assigned to the GPU that ranges from 50% up to 62%, in agreement with the improvement of the GPU performance with the problem size. Finally, the reduction operation of the dot makes the CPU more suitable for the smaller vectors, although with the growth of the problem size the relative cost of the reduction falls and the GPU rapidly increases its load ratio.

[1] We understand by thread divergence in the GPU when threads in the same block follow different execution paths. In stream processing such behavior results in a serial execution that degrades the performance.

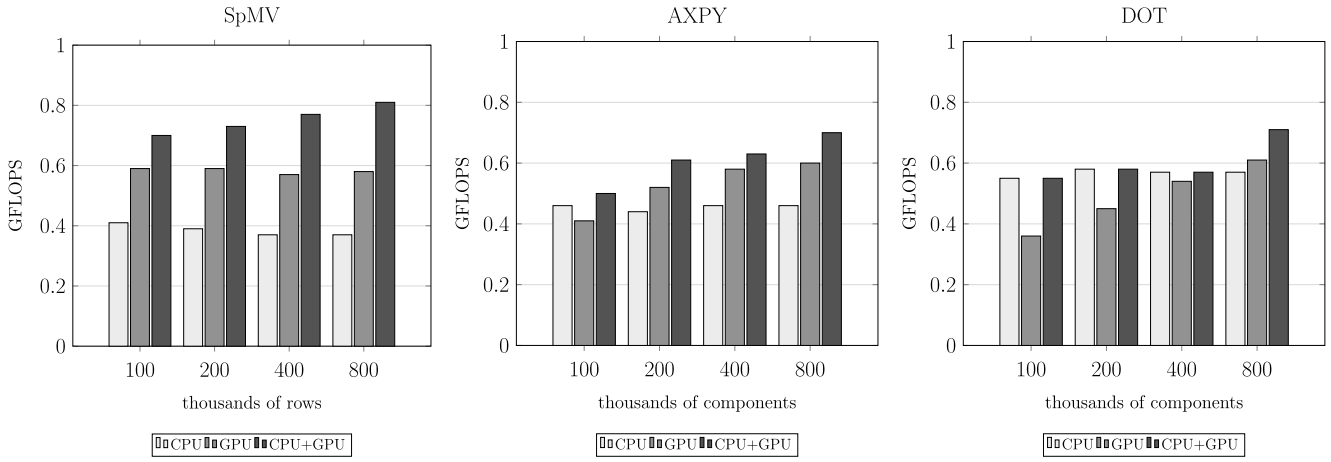

Fig. 7. Performance of the SpMV, AXPY and dot operations.

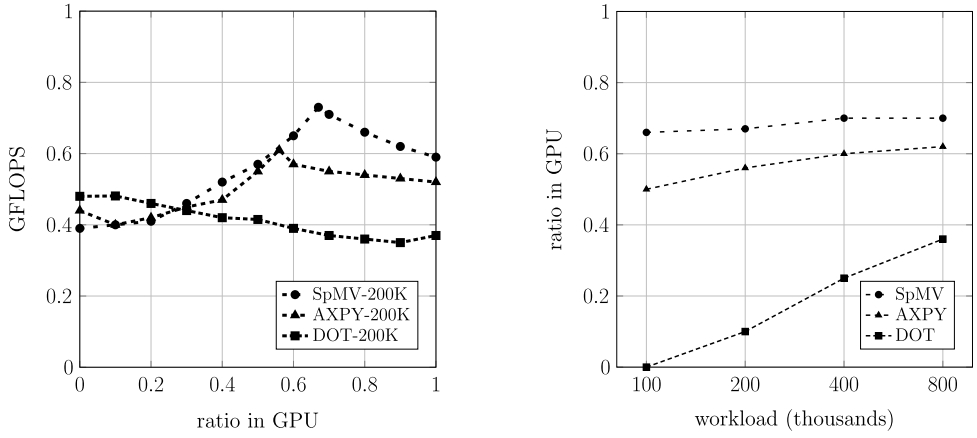

Fig. 8. Performance for different load distributions, problem with 200K unknowns (left) and balance point obtained for different problems sizes (right).

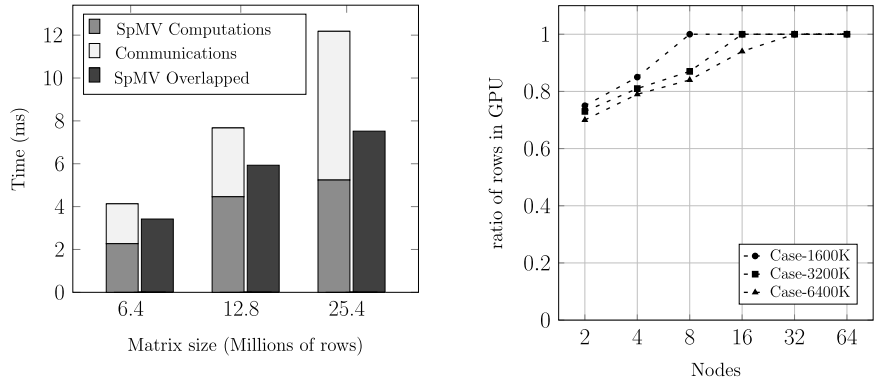

Fig. 9. Comparison of the overlapped and non-overlapped versions of the SpMV on 128 Mont-Blanc nodes (left) and balance point for the parallel execution of the overlapped SpMV (right).

5.3. Inter node performance: Overlapping

In the overlapped version of SpMV communications take place simultaneously with computations for the inner part of the matrix. In Fig. 9 (left) the overlapped and non-overlapped versions are compared on 128 Mont-Blanc nodes for mesh sizes from 6.4M up to 25.4M cells. For the non-overlapped cases the communication and computation costs are distinguished. The larger is the problem size the more communication costs are hidden. In these three examples the overlapping hides 37.79%, 54.19% and 67.19% of the communication costs, respectively. Consequently the overall SpMV time is reduced by 1.2×, 1.3× and 1.6×, respectively. Note that 128 Mont-Blanc nodes gather 256 CPU-cores and 128 GPUs.

Since the communication and computation phases are performed simultaneously, the load balancing of the kernel must include both phases too. This is not necessary for the dot product because the reduction communication is performed once the calculations are finished. The balance point is evaluated using the same Tabu search algorithm as in the sequential case. The communications are performed through the CPU, consequently a larger

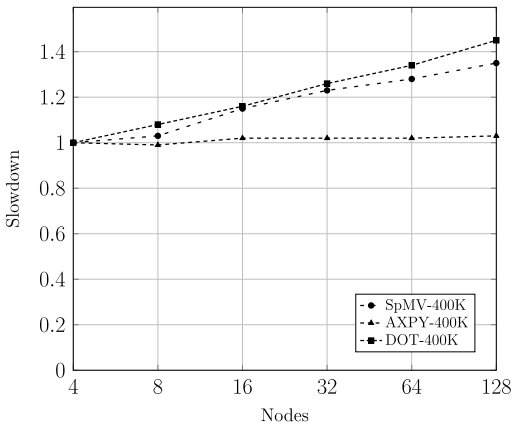

Fig. 10. Weak speedup of the algebraic kernels: SpMV, AXPY and dot.

workload is assigned to the GPU. Fig. 9 (right) shows the percentage of rows assigned to the GPU for different problem sizes and number of nodes. Both at decreasing the problem size or increasing the number of nodes, the relative weight of the communication costs grow, therefore the GPU increases its ratio. At a certain point the GPU computes the entire inner matrix because the communication costs are equal or larger than the computation costs, in the latter case the GPU will remain idle while the communication is completed. According to the plot, this situation occurs when the load per node is $\leq$ 200K.

5.4. Inter node performance: Scalability

The scalability of our code has been tested on the Mont-Blanc prototype in terms of both weak and strong scaling. In a weak scaling test the problem size grows proportionally to the number of computer nodes engaged so the workload per node remains constant (and the execution time should also be constant in the ideal case). Results of such a test are shown in terms of slowdown compared to the initial number of nodes. In contrast, in a strong scaling test the problem size remains constant when the number of nodes is increased gradually. In the ideal case execution time reduces proportionally to the number of nodes engaged.

Weak scaling tests for the three main kernels are shown in Fig. 10. The local workload of 400K cells per node is kept constant, the number of nodes grows from 4 to 128 while the total mesh size grows proportionally from 1.6M to 51.2M cells. As expected, AXPY shows a perfect scaling due to the absence of communications. In contrast, the dot product and the SpMV show slowdowns of 1.45× and 1.35×, respectively.

The strong scaling results are shown in terms of the speedup compared to the initial number of nodes and the parallel efficiency (in percents of the ideal speedup). In Fig. 11 (left), the parallel efficiency of the SpMV is shown for different mesh sizes. Certainly, the larger is the size of the problem the higher is the speedup achieved, because the relative weight of the communications is inversely proportional to the problem size. In Fig. 11 (right), the strong speedup of the main kernels is studied for a mesh of 12.8M cells used in the LES simulation of the flow around ASMO car [28]. An estimation of the overall code performance is also included. This estimation can be done from the performance of the main kernels, knowing the repetitions of each one per time step. In [18] is shown its good accuracy for different applications. Note that despite the AXPY operation does not require communications, its strong scalability is not linear. The reason for this behavior is the reduction of the GPU occupancy that at certain point degrades its performance. This effect can be clearly observed in Fig. 7. For the dot product this degradation factor is added to the increment of the communication costs with the number of nodes. Note that its parallel efficiency is below 25% on 128 nodes. Finally the parallel efficiency of the SpMV and of the overall simulation are very similar (around 50% on 128 nodes) since the SpMV is the dominant kernel as shown in Fig. 1. Note that the interconnection technology based on Ethernet limits the bandwidth to 1 Gb/s. Moreover, the use of mobile SoC missing PCIe links, forced the Mont-Blanc consortium to bridge USB3 to Ethernet. This design compromise increases the latency of the network as analyzed in [12]. Significant improvements are expected on these aspects in the next Mont-Blanc prototype which will be based on the Cavium ThunderX2 processor [3]. This high-end ARM-based SoC is targeting server market and will feature several PCIe lanes.

5.5. Comparative analysis: Performance and energy efficiency

We compare the performance and energy efficiency of Mont-Blanc nodes with respect to the heterogeneous nodes of the MinoTauro supercomputer from BSC. MinoTauro nodes have two NVIDIA M2090 GPUs and two 6-core Intel Xeon E5649 CPUs, and are interconnected with an InfiniBand QDR network. The version of code for MinoTauro is based on the same operational approach as used in Mont-Blanc, but implemented by means of MPI+CUDA [18]. This version features similar optimizations such as reordering of unknowns, overlapping and adapted storage formats, but not the heterogeneous mode. The benefit of using CPUs for computations on MinoTauro is relatively small. Firstly, it is because the GPUs are many times faster than the CPUs. Secondly, distribution of workload between CPU and GPU results in additional intra-node traffic and, as a consequence, notably increases overhead of the host-device transfers. It must also be noted, that CUDA kernels are nearly identical to the OpenCL ones and both versions perform nearly equal on NVIDIA GPUs.

Fig. 12 (left) shows comparison of the performance on both systems for the simulation of the flow around the ASMO car (Re $= 7 \times 10^5$) on a mesh of 5.5 million cells. The performance is measured in GFLOPS per node and corresponds to the time-integration phase. The time is obtained by averaging measurements over 10,000 time steps.

The proposed adapted implementation outperforms the baseline version by around 35% on average. The most notable benefit (up to 53%) is observed at the smaller number of nodes because the low weight of communications leaves more time for the CPU to compute.

The performance on MinoTauro is on average 33× higher than on Mont-Blanc. This difference mainly comes from the much bigger computing power and memory bandwidth of the MinoTauro computing devices and from the much more powerful interconnect. In both systems, the performance in GFLOPS per node reduces when the number of nodes grows. It is because, firstly, the GPUs performance goes down at smaller workloads and, secondly, the communication costs increase. This effect is more severe on MinoTauro, consequently its speedup compared to Mont-Blanc decreases with the number of nodes, see Fig. 12 (right).

A detailed on-line measurement of the power consumption was not possible on MinoTauro for the present study. However, an estimation can be obtained based on the idle power (280 W) and the peak power (800 W) of a single node of MinoTauro. According to the previous measurements performed on Mont-Blanc system [12], in idle state a Mont-Blanc node consumes 5 W, increasing up to 11 W when running the LINPACK benchmark. Since our application is mainly constrained by the memory accesses, less than 3% of the peak performance is achievable in both systems. In particular for a single node execution the percentages achieved are 2.2% and 2.8% on MinoTauro and Mont-Blanc, respectively.

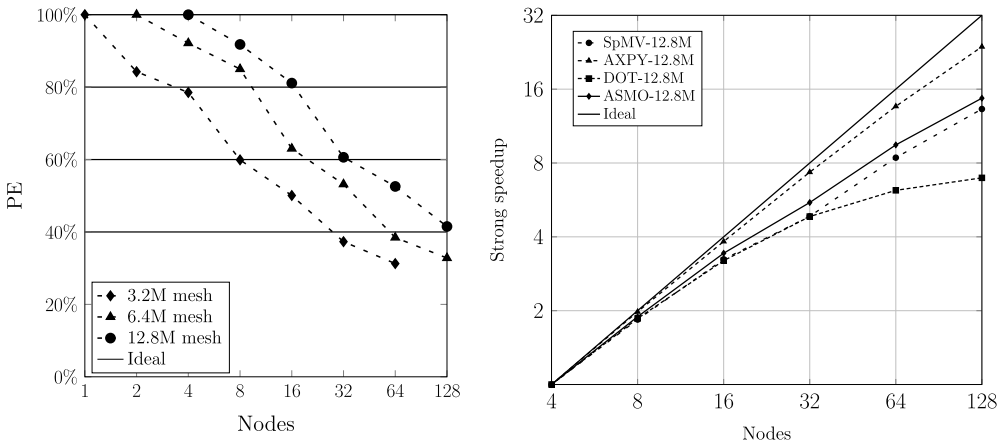

Fig. 11. Parallel efficiency for the overlapped SpMV (left) and the strong speedup of the main kernels and of the overall code (right).

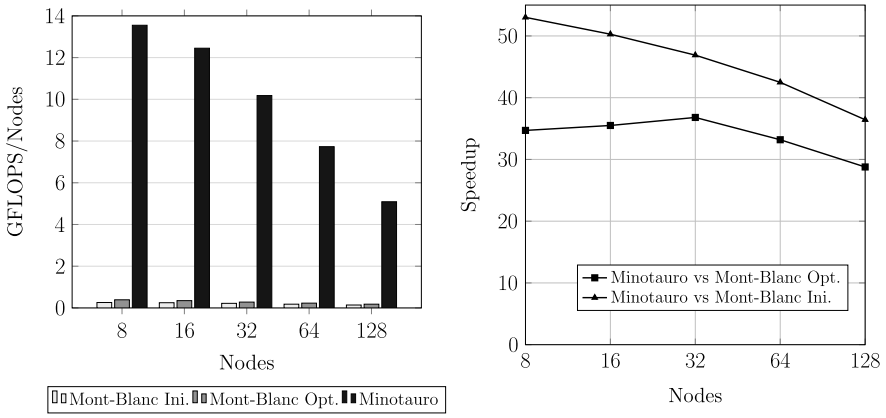

Fig. 12. GFLOPS per node achieved in the simulation of the flow around ASMO car for different number of nodes (left) and the speedup of MinoTauro vs Mont-Blanc nodes (right).

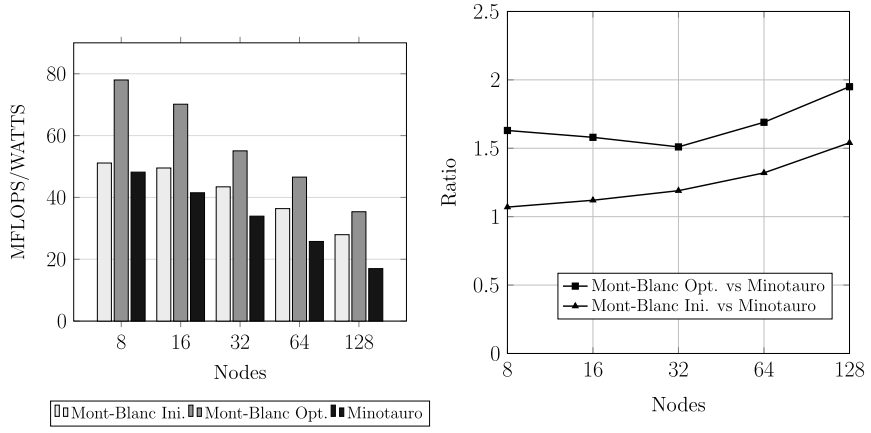

Fig. 13. MFLOPS-per-Watt in the simulation of the flow around ASMO car for different number of nodes (left) and ratio of Mont-Blanc vs MinoTauro in this context (right).

Consequently we can assume that the power consumption will be close to the idle state, and therefore utilize it in our estimation. The FLOPS-per-Watt for the different executions of the ASMO car test are illustrated in Fig. 13 (left). Additionally, the energy efficiency of Mont-Blanc vs MinoTauro is shown in Fig. 13 (right). In average the Mont-Blanc nodes are 1.7× more energy-efficient than Minotauro's nodes. If we had used the peak power consumption of both nodes as consumption estimation this ratio would be 2.2×. We can therefore expect that the real power efficiency gain ranges between these two values. Real measurements of the energy costs of Mont-Blanc nodes compared to MareNostrum III supercomputer from BSC can be found in [12]. In this study improvements of 1.7× are observed, thus supporting the estimation introduced in our current tests.

6. Conclusions

The challenge of building a supercomputer that can deliver and sustain Exascale performance at acceptable power consumption

costs ($\approx$ 20MW) has stressed the research on new computing architectures. In this context, Mont-Blanc project is the main European initiative focused on leveraging the huge investments of the exponentially growing market of mobile devices to build new HPC systems. In this paper, we have described the optimizations performed to TermoFluids CFD code to attain the maximum performance in this new HPC architecture.

Our first concern has been to fully occupy the intra node resources. We have developed a hybrid version of the main kernels that compose our CFD code in order to engage both the CPU and GPU devices of each node. This has been achieved by means of an OpenCL+OpenMP implementation with optimized storage formats for each device. Moreover, we have included a dynamic load balance algorithm based on a Tabu search method to efficiently distribute the workload. The best results of this approach were obtained for the SpMV, which happens to be the most time consuming operation of our code. In this operation, up to 2.1× speedup was obtained with respect to the CPU-only execution. The remaining operations, AXPY and DOT, attained speedups of 1.3× and 1.24× respectively.

The following step has been to optimize the communication scheme in order to improve the parallel efficiency of the algorithm. We have adopted an overlapping strategy for the SpMV in which communications and computations are performed concurrently. Following this approach, up to 67% of the communication costs are hidden for a mesh of 25.4 million cells. The same Tabu search balance algorithm is used for the workload distribution in parallel. Since the communications are performed by the CPU, the new distribution assigns more load to the GPU. In addition, scalability tests are provided engaging up to 128 Mont-Blanc nodes. The weak speedup tests show that for a reasonable workload of 400K cells, the maximum slowdown of the operations is 1.45×. The strong speedup tests show a parallel efficiency of about 50% at increasing the number of Mont-Blanc nodes from 4 up to 128. Altogether the new implementation of the CFD algorithm demonstrates on the Mont-Blanc system an improvement of 35% on average with respect to the baseline MPI-only version for CPUs. Finally, we have presented a comparative analysis of the performance and power consumption versus the high-end heterogeneous supercomputer MinoTauro. We have estimated that Mont-Blanc nodes are about 30 times slower but 41% more energy efficient at running a LES simulation of the flow around ASMO car.

The future generation of Mont-Blanc prototypes are planned to be built with high-end ARM technology that has already been proven in data centers and cloud computing. Such technology upgrade will increase the overall system performance, reducing in special the network limitations. This paper has presented some algorithmic and implementation strategies that can be useful to unlock the potential of these incoming generation of ARM-based systems.

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007–2013] and Horizon 2020 under the Mont-Blanc Project (www.montblanc-project.eu), grant agreement n 288777, 610402 and 671697. The work has been financially supported by the Ministerio de Ciencia e Innovación, Spain (ENE- 2014-60577-R), the Russian Science Foundation, project 15-11-30039, CONICYT Becas Chile Doctorado 2012, the Juan de la Cierva posdoctoral grant (IJCI-2014-21034), and the Initial Training Network SEDITRANS (GA number: 607394), implemented within the 7th Framework Programme of the European Commission under call FP7-PEOPLE-2013-ITN. Our calculations have been performed on the resources of the Barcelona Supercomputing Center. The authors thankfully acknowledge these institutions.

References

[1] D.E. Keyes, Exaflop/s: the why and the how, C. R. Méc. 339 (2–3) (2011) 70–77. http://dx.doi.org/10.1016/j.crme.2010.11.002.
[2] J. Dongarra, et al., The international exascale software project roadmap, Int. J. High Perform. Comput. Appl. 25 (1) (2011) 3–60. http://dx.doi.org/10.1177/1094342010391989.
[3] Mont-Blanc project. http://www.montblanc-project.eu/.
[4] GW4 Alliance, Isambard ARM-based supercomputer. http://gw4ac.uk/isambard/..
[5] L. Natvig, T. Follan, S. Stoa, S. Magnussen, A. García Guirado, Limbing Mont Blanc - A Training Site for Energy Efficient Programming on Heterogeneous Multicore Processors, 2015. CoRR abs http://1511.02240.
[6] N. Rajovic, A. Rico, N. Puzovic, C. Adeniyi-Jones, A. Ramirez, Tibidabo: Making the case for an ARM-based HPC system, Future Gener. Comput. Syst. (2013).
[7] N. Rajovic, A. Rico, J. Vipond, I. Gelado, N. Puzovic, A. Ramirez, Experiences with mobile processors for energy efficient HPC, in: Design, Automation and Test in Europe Conference and Exhibition, DATE, 18–22 March 2013, 2013, pp. 464, 468.
[8] N. Rajovic, P.M. Carpenter, I. Gelado, N. Puzovic, A. Ramirez, M. Valero, Supercomputing with commodity CPUs: are mobile SoCs ready for HPC? in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '13, ACM, New York, NY, USA, 2013. Article 40 , 12 pages.
[9] K.P. Saravanan, P.M. Carpenter, A. Ramirez, A performance perspective on energy efficient HPC links, in: Proceedings of the 28th ACM International Conference on Supercomputing, ICS '14, ACM, New York, NY, USA, 2014, pp. 313–322.
[10] D. Göddeke, D. Komatitsch, M. Geveler, D. Ribbrock, N. Rajovic, N. Puzovic, A. Ramirez, Energy efficiency vs. performance of the numerical solution of PDEs: An application study on a low-power ARM-based cluster, J. Comput. Phys. (ISSN: 0021-9991) 237 (2013) 132–150.
[11] E. Calore, S.F. Schifano, R. Tripiccione, Energy-performance tradeoffs for HPC applications on low power processors, in: S. Hunold, A. Costan, D. Giménez, A. Iosup, L. Ricci, M.E.G. Requena, V. Scarano, A.L. Varbanescu, S.L. Scott, S. Lankes, J. Weidendorfer, M. Alexander (Eds.), Euro-Par 2015: Parallel Processing Workshops, Springer International Publishing, 2015, pp. 737–748.
[12] Nikola Rajovic, et al., The mont-blanc prototype: an alternative approach for hpc systems, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '16, IEEE Press, Piscataway, NJ, USA, 2016 Article 38 , 12 pages.
[13] I. Grasso, P. Radojkovic, N. Rajovic, I. Gelado, A. Ramirez, Energy efficient HPC on embedded SoCs: Optimization techniques for mali GPU, in: Parallel and Distributed Processing Symposium, 2014 IEEE 28th International, 19–23 May 2014, pp. 123, 132.
[14] J.A. Ross, D.A. Richie, S.J. Park, D.R. Shires, L.L. Pollock, A case study of OpenCL on an android mobile GPU, in: High Performance Extreme Computing Conference, HPEC, IEEE, 2014 pp. 1, 6, 9–11.
[15] O. Lehmkuhl, C.D. Perez-Segarra, R. Borrell, M. Soria, A. Oliva, Termofluids: A new parallel unstructured CFD code for the simulation of turbulent industrial problems on low cost PC cluster, Parallel CFD 2007, Lect. Notes Comput. Sci. Eng. 67 (2008) 275–282.
[16] R. Borrell, J. Chiva, O. Lehmkuhl, G. Oyarzun, I. Rodríguez, A. Oliva, Optimising the termofluids CFD code for petascale simulations, Int. J. Comput. Fluid Dyn. 30 (6) (2016).
[17] J. Dongarra, Basic linear algebra subprograms technical forum standard, Int. J. High Perform. Appl. Supercomput. 16 (1) (2002) 1–111; Int. J. High Perform. Appl. Supercomput. 16 (2) (2002) 115–199.
[18] G. Oyarzun, Heterogeneous Parallel Algorithms for Computational Fluid Dynamics on Unstructured Meshes (Ph.D. thesis), Polytechnic University of Catalonia, 2015 (Chapter 2). www.tdx.cat/handle/10803/323892.
[19] F.X. Trias, O. Lehmkuhl, A. Oliva, C.D. Perez-Segarra, R.W.C.P. Verstappen, Symmetry-preserving discretization of Navier–Stokes equations on collocated unstructured grids, J. Comput. Phys. 258 (2014) 246–267.
[20] A.J. Chorin, Numerical solution of the Navier–Stokes Equations, J. Comput. Phys. 22 (1968) 745–762.
[21] A. Monakov, A. Lokhmotov, A. Avetisyan, Automatically tuning sparse matrix-vector multiplication for GPU architectures, high performance, embedded architectures and compilers, Lecture Notes in Comput. Sci. 5952 (2010) 111–125.
[22] G. Oyarzun, R. Borrell, A. Gorobets, A. Oliva, MPI-CUDA sparse matrix–vector multiplication for the conjugate gradient method with an approximate inverse preconditioner, Comput. & Fluids 92 (2014) 244–252.
[23] D.B. Heras, V. Blanco, J.C. Cabaleiro, F.F. Rivera, Modeling and improving locality for the sparse-matrix–vector product on cache memories, Future Gener. Comput. Syst. 18 (1) (2001) 55–67.
[24] E. Cuthill, J. McKee, Reducing the bandwidth of sparse symmetric matrices, in: Proc. 24th Nat. Conf, ACM, 1969, pp. 157–172.

[25] C. Gil, J. Ortega, A.F. Díaz, M.D.G. Montoya, Annealing-based heuristics and genetic algorithms for circuit partitioning in parallel test generation, Future Gener. Comput. Syst. 14 (5) (1998) 439–451.
[26] W. Yang, K. Li, K. Li, A hybrid computing method of SpMV on CPU–GPU heterogeneous computing systems, J. Parallel Distrib. Comput. (2017).
[27] W. Yang, K. Li, Z. Mo, K. Li, Performance optimization using partitioned SpMV on GPUs and multicore CPUs, IEEE Trans. Comput. 64 (9) (2015) 2623–2636.
[28] D.E. Aljure, O. Lehmkuhl, I. Rodríguez, A. Oliva, A flow and turbulent structures around simplified car models, Comput. & Fluids 96 (2014) 122–135.

Dr. Guillermo Oyarzun is a Marie Curie post-doc researcher of the SEDITRANS project in the Department of Civil Engineering at the University of Patras, Greece. He obtained his Ph.D. at Barcelona Tech, Spain. He currently collaborates as external researcher in the Heat and Mass Transfer Technological Center (CTTC) and in Termo Fluids S.L. His scientific activity has been developed in the fields of Computer Science applied to Computational Fluid Dynamics, with a focus on the development of new numerical methods and software tools for the emerging technologies in High Performance Computing (HPC).

Dr. Ricard Borrell is a Juan de la Cierva postdoctoral researcher at the Barcelona Supercomputing Center (BSC). His research is conducted in the areas of Applied Mathematics and Computer Science for Computational Mechanics, with the focus on High Performance Computing. He is author of 17 journal papers and of more than 75 contributions to international conferences. Additionally, he is co-founder of Termo Fluids S.L (www.termofluids.com) a spinoff of Barcelona Tech, aiming to transfer academic knowledge and experience to the industrial sector. Moreover, he has been member of the Industrial Advisory Committee of PRACE for two years.

Dr. Andrey Gorobets is a leading researcher at the Keldysh institute of applied mathematics (KIAM) of Russian academy of sciences (RAS), Moscow, Russia. He is twice Ph.D., one in the Institute for math modeling of RAS, Russia, one in Barcelona Tech, Spain, and once Dr. Sc. at KIAM. All the degrees are in high performance computing of computational fluid dynamics problems. As a result, He has 41 journal publications and He has participated in 89 conference talks on this subject. His main research topic now is development of parallel algorithms for large-scale numerical simulations on hybrid supercomputers.

Dr. Filippo Mantovani is a postdoctoral research associate of the Mobile and embedded-based HPC group at the Barcelona Supercomputing Center (BSC). He graduated in mathematics and holds a Ph.D. in Computer Science from University of Ferrara, Italy. He has been a scientific associate at the DESY laboratory in Zeuthen, Germany, and at the University of Regensburg, Germany. He spent most of his scientific career in computational physics, computer architecture and high-performance computing, contributing to the Janus, QPACE and QPACE2 projects. He joined BSCs Mont-Blanc project in 2013, becoming in 2014 principal investigator of the project.

Professor Assensi Oliva Founder and head of the Heat and Mass Transfer Technological Center (CTTC), one of the leading research centers in the field of Turbulence and Heat and Mass transfer. He has more than 40 years of experience working on heat and mass transfer phenomena analysis, fluid mechanics (CFD), numerical simulation models and experimental test facilities. As a result of his research activity, he has been author of more than 100 publications in international journals of mechanical and thermal engineering in the last 15 years and about 300 contributions in peer-review international conferences.